Personalized Transaction Kernels for Recommendation using MCTS

Maryam Tavakol[1,2⋆], Tobias Joppen[3⋆], Ulf Brefeld[1], and Johannes Fürnkranz[3]

[1] Machine Learning group, Leuphana Universität Lüneburg, Lüneburg, Germany
brefeld@leuphana.de
[2] Artificial Intelligence group, TU Dortmund, Dortmund, Germany
maryam.tavakol@cs.tu-dortmund.de
[3] Knowledge Engineering group, TU Darmstadt, Darmstadt, Germany
{tjoppen,juffi}@ke.tu-darmstadt.de

Abstract. We study pairwise preference data to model the behavior of users in online recommendation problems. We first propose a tensor kernel to model contextual transactions of a user in a joint feature space. The representation is extended to all users via hash functions that allow to effectively store and retrieve personalized slices of data and context. In order to quickly focus on the relevant properties of the next item to display, we investigate the utility of Monte-Carlo Tree Search (MCTS) on the learned preference values. Empirically, on real-world transaction data, both the preference models as well as the search tree exhibit excellent performance over baseline approaches.

Keywords: Preference learning · Tensor kernel · Personalization · MCTS.

1 Introduction

Understanding user behavior is essential in many recommendation tasks involving implicit feedback. Several approaches have been aimed to capture characteristic traits of users by analyzing data, ranging from atomic user actions such as clicks and purchases to their entire navigation patterns. Nevertheless, for the vast majority of users, the available data is very limited. User clicks do not per se express an interest in an item, purchases are generally rare events, by definition, and a reliable analysis of navigation patterns requires regular (and possibly frequent) visits of the same user. Maintaining an individual model for every user bears another caveat; besides retrieval and maintenance costs, again only heavy hitters will really benefit from such an approach. Therefore, there is a great need for techniques that leverage all available data so that *every* user benefits, irrespectively of their amount of data.

In this paper, we explore pairwise preference data to study the behavior of users in online recommendation problems. We propose to use qualitative feedback in form of pairwise preferences as the lever. Preference data discloses the

⋆ Both authors contributed equally

true interests of a user and have often served as a reliable source of information [8, 10]. However, preferences highly depend on the context of the user. We use results from tensor theory to propose a transaction kernel that maps multiple data sources into a joint feature space (e.g., the user's click history, context, demographics, etc.). Hash functions augment data from other users into this space such that user slices can be efficiently stored and retrieved. The kernel can be extended to pairwise preference data and a preference learning algorithm such as SVM^{rank} [12] can be used to obtain personalized preference models.

Furthermore, we introduce an online search technique to recommend the most relevant items to the actual user and context. We thus benefit from the personalized preference models as a utility function to conduct such informed sampling in product recommendation tasks. The utility function needs to be maximized in order to identify the optimal item(s), i.e., those items that are most likely to be clicked on by the user in a given context according to the learned model. A naïve approach for computing the best item to recommend is exhaustive search over all possible items and return the one with the maximum value. An anytime version of this algorithm returns the product with the highest value among those that have been seen so far. To improve this, we employ a variant of Monte Carlo Tree Search (MCTS) which allows to quickly focus on items with desirable features. Our results show that the MCTS variant returns better recommendations in cases where the number of sampled products is limited. The main contribution of our paper is therefore presenting a highly efficient approach for modeling user behavior in online recommendation scenarios which is remarkably effective.

2 Related Work

Personalized recommender systems range from collaborative-based methods [20, 11] and matrix factorization [15], to contextual and session-based approaches [26, 29, 16]. Recommender systems leverage quantitative feedback either in form of explicit ratings or implicit views to retrieve items of interest to the user.

An alternative viewpoint constitutes scenarios that are based on feedback in form of user preferences [10]. Preference learning describes a family of learning problems where the target information is not necessarily given, but preferences between options are known; the task is to predict an ordering between these options [8]. One can distinguish between object ranking problems [13], and label ranking problems [28]. Both problems can be approached in different ways. We formalize our problem as an object ranking task, that we address by learning an underlying utility function. We learn a personalized preference model using SVMs, in a similar fashion to Joachims [12], who effectively utilizes an SVM to learn a ranking function for click-through data, and Chapelle and Keerthi [5] who present an efficient method to speed up the algorithm. Furthermore, the use of SVMs facilitates to deal with non-linearity by using the kernel trick. Kernel methods have been successfully employed for top-N recommendation using, for instance, Gaussian processes [27] or contextual bandits [25].

In this paper, we benefit from tensor kernels to express the conjugation of different feature representations (or contexts) by tensor products. Tensor products [7], both as an explicit feature mapping and kernel function, have been employed for feature selection in classification tasks [3, 4, 24]. Oyama and Manning [18] propose a tensor kernel to conjugate features of example pairs for learning pairwise classifiers. Tensors are additionally used for relation extraction in unstructured natural language parsing [32]. The idea of joint feature maps using tensor product is further utilized in recommendation, where Basilico and Hofmann [1] present a collaborative-based kernel method over user-item pairs for rating prediction. Instead, we use hash functions for learning user-specific models, and empirically show that our approach significantly outperforms their algorithm yet with a much faster computation.

Hashing functions are introduced by Shi et al. [22] for sparse projections in multi-class classification tasks, and are originally known as *Count Sketch* [6]. Weinberger et al. [31] propose a hashing trick for large-scale multi-task learning, where all tasks are mapped into a joint hash space. Together with tensor products, Pham and Pagh [19] apply hashing as a random feature mapping to approximate polynomial kernels in large-scale problems with bounded error. They exhibit the tensor product feature space as an equivalent to polynomial kernels and propose an efficient way to project the data into a lower dimension without explicitly computing the tensor products. Subsequently, Wang et al. [30] exploit randomized tensors to efficiently perform implicit tensor decomposition for latent topic modeling.

Our learned model is used in an MCTS framework to sample near-optimal products for online recommendation. MCTS is an anytime tree search algorithm for sequential decision making [14, 2] which became very popular due to the great success in the game playing domains such as Go [23]. MCTS has been also employed in recommendation problems. Liebman et al. [17] propose MCTS for playlist recommendation and develops alternative backup strategies to increase convergence speed. Moreover, Gaudel and Sebag [9] deploy MCTS in feature selection for recommendation tasks.

3 Informed Sampling from Personalized Preferences

3.1 Preliminaries

We study transaction scenarios in which users, represented by their user ID $u \in \mathbb{U}$, click on a product $\mathbf{p} \in \mathbb{P}$. The context of the click (e.g., the sequence of previous clicks, the day and time, etc.) is captured by $\mathbf{s} \in \mathbb{S}$. We aim to understand why user u in context $\mathbf{s}$ clicks on item $\mathbf{p}$ and not on some other presented item $\mathbf{p}'$ and to turn this understanding into a recommender system that shows interesting new items to users depending on their context.

Before we exploit user preferences, note that there is often more information available than the triplet of user ID, item, and context. Some scenarios may provide additional data sources such as user profile data $\mathbb{A}$, shipping and billing

addresses $\mathbb{B}$, additional information on items $\mathbb{I}$, user friendship graphs $\mathbb{F}$, or further demographics $\mathbb{D}$. Without loss of generality, we thus assume the existence of m different data sources, $\mathcal{X} = \{\mathbb{X}^1, \ldots, \mathbb{X}^m\}$ where every source adds some pieces of information to the problem. We first consider the generalized problem of merging the m sources into a personalized joint representation before we incorporate preferences and learn a utility function that can be used together with Monte-Carlo tree search-based approaches.

3.2 Transaction Kernels

Tensor Kernels. In order to completely capture the interlace properties of data sources, $\{\mathbb{X}^1, \ldots, \mathbb{X}^m\}$, we define the mapping ψ^t as the tensor product of their respective vector spaces. The tensor product of two vector spaces V and V' is again a vector space $V \otimes V'$ [7]. Let $\mathbf{v} = \{v_1, v_2, \ldots, v_k\}$ and $\mathbf{v}' = \{v'_1, v'_2, \ldots, v'_d\}$ be the basis systems of V and V', respectively. Their tensor product space is spanned by a basis that contains all pairs (v_i, v'_j). For instance, if $\mathbf{v} = \{v_1, v_2, v_3\}$ and $\mathbf{v}' = \{v'_1, v'_2\}$, the tensor product $\mathbf{v} \otimes \mathbf{v}'$ is $\{v_1v'_1, v_1v'_2, v_2v'_1, v_2v'_2, v_3v'_1, v_3v'_2\}$. Applying this to our setting results in a mapping ψ^t on $\mathbf{x} \in \mathcal{X}$ which is given by

$$\psi^t(\mathbf{x}) = \psi^t(\mathbf{x}^1, \ldots, \mathbf{x}^m) = \mathbf{x}^1 \otimes \ldots \otimes \mathbf{x}^m. \tag{1}$$

Let $n_1, \ldots, n_m$ be the dimensions of the feature spaces, $\forall \mathbf{x}, \mathbf{z} \in \mathcal{X}$ we derive

$$\begin{aligned}
\langle \psi^t(\mathbf{x}), \psi^t(\mathbf{z}) \rangle &= \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \ldots \sum_{l=1}^{n_m} (x_i^1 x_j^2 \ldots x_l^m)(z_i^1 z_j^2 \ldots z_l^m) \\
&= \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \ldots \sum_{l=1}^{n_m} x_i^1 z_i^1 x_j^2 z_j^2 \ldots x_l^m z_l^m \\
&= \langle \mathbf{x}^1, \mathbf{z}^1 \rangle \langle \mathbf{x}^2, \mathbf{z}^2 \rangle \ldots \langle \mathbf{x}^m, \mathbf{z}^m \rangle.
\end{aligned}$$

Thus, the *tensor kernel* k^t is obtained by multiplying the corresponding inner products between the spaces

$$k^t(\mathbf{x}, \mathbf{z}) = \prod_{r=1}^{m} \langle \mathbf{x}^r, \mathbf{z}^r \rangle. \tag{2}$$

The tensor product features are equivalent to the product of kernels for all domains in case of linear kernel [24]. Note that the proposed kernel possesses an explicit representation given in Equation (1) that may be useful in large-scale tasks with small dimensionalities.

Personalized Kernels. As we mentioned in Section 3.1, we consider the user IDs a source of information. However, tensor products of such terms with other views act as normal inner products that do not affect the learning process in a meaningful way. We thus propose a hashed feature mapping ψ^p on top of the

Table 1. Exemplary feature mappings

Collective	Personalized	Personalized+Collective
red::white		red::white
red::women		red::women
red::shoes		red::shoes
...		...
Adidas::sneakers		Adidas::sneakers
Adidas::Nike		Adidas::Nike
	u-742::red::white	u-742::red::white
	u-742::red::women	u-742::red::women
	u-742::red::shoes	u-742::red::shoes
	...	...
	u-742::Adidas::sneakers	u-742::Adidas::sneakers
	u-742::Adidas::Nike	u-742::Adidas::Nike

tensor product ψ^t to remedy this limitation. Given two hash functions $g : \mathbb{N} \to \{1, \ldots, d\}$ and $\gamma : \mathbb{N} \to \{-1, +1\}$ the hashed feature map ψ^p is defined as

$$\psi_i^p(\mathbf{x}) = \sum_{j:g(j)=i} \gamma(j) x_j \qquad \text{see [31]}, \tag{3}$$

where d is the hash size and the binary hash γ is used to remove the bias inherent in the hash kernel. Consequently, the obtained hashing function gives rise to the *personalized kernel* k^p

$$k^p(\mathbf{x}, \mathbf{z}) := \langle \psi^p(\psi^t(\mathbf{x})), \psi^p(\psi^t(\mathbf{z})) \rangle. \tag{4}$$

The presence of a user ID automatically leads to a user-specific representation without the need to maintain an individual model for every user. Hence, the personalized kernel individually hashes all data sources into user slices and allows to control the dimensionality of the resulting feature space via the number of bits in the hash function. Moreover, the length of the hashed vector is preserved with high probability in the new space [31].

Collective Kernels. A substantial problem of personalized systems is to cope with cold start situations. Usually, many users in the system have no or too few transactions, which leads to inaccurate personalized models. Borrowing ideas from Tavakol and Brefeld [25] and Weinberger et al. [31], we propose an additional collective kernel that stores all user data in a single slice to account for users and contexts with limited data. Therefore, the *collective kernel* function k^c simply discards the user IDs from the tensor products and is thus given by

$$k^c(\mathbf{x}, \mathbf{z}) := \langle \psi^p(\psi^t(\mathbf{x} \setminus u)), \psi^p(\psi^t(\mathbf{z} \setminus u)) \rangle. \tag{5}$$

Combining Personalized and Collective Kernels. We propose to combine the personalized and the collective kernels into a single kernel function to have the best of the two worlds. Every user has their own individual model with all data created by that user and whenever that information is insufficient, the collective part of the kernel may help out. Given the union operation $\cup$, the combined feature map ψ^{pc} is given by

$$\psi^{pc}(\mathbf{x}) = \psi^{p}\Big(\psi^{t}(\mathbf{x}) \cup \psi^{t}(\mathbf{x} \setminus u)\Big), \tag{6}$$

and leads to the *personalized and collective kernel* k^{pc},

$$k^{pc}(\mathbf{x}, \mathbf{z}) := \langle \psi^{pc}(\mathbf{x}), \psi^{pc}(\mathbf{z}) \rangle. \tag{7}$$

As a result, three models are considered: a collective model, a personalized model, and a personalized+collective model. Note that the former learns the same parameters for all the users. To shed light on the characteristic traits of the proposed kernels, we showcase their feature spaces on the example of a user `u-742` with context $\mathbf{s}$ = `red`, `women`, `shoes`, `sneakers`, `Adidas` currently viewing an item $\mathbf{p}$ = `white`, `women`, `shoes`, `sneakers`, `Nike`. Table 1 shows the resulting features for the collective, personalized, and personalized+collective feature maps where the tensor product is represented as the concatenation of the corresponding features. Note that we ignore the hash functions for a moment, which would map the resulting strings to numbers.

Preference-based Transaction Kernels. Finally, to leverage pairwise preference data for the recommendation scenarios, we consider user preferences of the form $\{\mathbf{p}_i \succ \mathbf{p}'_i \mid u_i, \mathbf{s}_i\}_{i=1}^{n}$, indicating that user u_i prefers item $\mathbf{p}_i$ over $\mathbf{p}'_i$ in context $\mathbf{s}_i$. Every preference is translated into $\mathbf{x}_i = (\mathbf{p}_i, u_i, \mathbf{s}_i)$ and $\mathbf{x}'_i = (\mathbf{p}'_i, u_i, \mathbf{s}_i)$. Note that additionally available data sources are simply appended in the representation. Using a linear model with parameters $\boldsymbol{\theta}$, we require that $\boldsymbol{\theta}^{\top}\psi^{pc}(\mathbf{x}_i) \geq \boldsymbol{\theta}^{\top}\psi^{pc}(\mathbf{x}'_i)$. Due to linearity, we have

$$\boldsymbol{\theta}^{\top}\left(\psi^{pc}(\mathbf{x}_i) - \psi^{pc}(\mathbf{x}'_i)\right) \geq 0. \tag{8}$$

After certain transformations, the representer theorem [21] allows to rewrite the primal parameters as

$$\boldsymbol{\theta} = \sum_{j} \alpha_j \left(\psi^{pc}(\mathbf{x}_j) - \psi^{pc}(\mathbf{x}'_j)\right)$$

for dual variables α_j. Plugging this result back into Equation (8) shows that all data-driven parts are of the form

$$\left\langle \psi^{pc}(\mathbf{x}_i) - \psi^{pc}(\mathbf{x}'_i), \psi^{pc}(\mathbf{x}_j) - \psi^{pc}(\mathbf{x}'_j) \right\rangle.$$

Expanding the term gives

$$\langle \psi^{pc}(\mathbf{x}_i), \psi^{pc}(\mathbf{x}_j) \rangle - \langle \psi^{pc}(\mathbf{x}_i), \psi^{pc}(\mathbf{x}'_j) \rangle - \langle \psi^{pc}(\mathbf{x}'_i), \psi^{pc}(\mathbf{x}_j) \rangle + \langle \psi^{pc}(\mathbf{x}'_i), \psi^{pc}(\mathbf{x}'_j) \rangle,$$

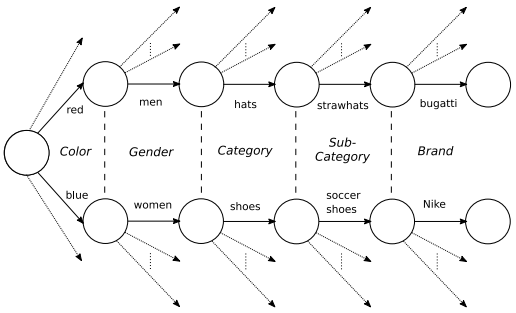


Fig. 1. The structure of search tree used by MCTS with five categorical layers.

and using Equation (7) leads to the desired *preference-based transaction kernel* that is given by

$$k(\mathbf{x}_i \succ \mathbf{x}'_i, \mathbf{x}_j \succ \mathbf{x}'_j) = k^{pc}(\mathbf{x}_i, \mathbf{x}_j) - k^{pc}(\mathbf{x}_i, \mathbf{x}'_j) - k^{pc}(\mathbf{x}'_i, \mathbf{x}_j) + k^{pc}(\mathbf{x}'_i, \mathbf{x}'_j).$$

Note that in the experiments, we also evaluate the other proposed kernels, k^p and k^c, as well. The kernel can be plugged into a binary support vector machine or any other kernel machine. In case of the former, every preference encodes a positive example which renders the problem a binary ranking task. Note that thresholds cancel out in Equation (8); hence, the optimal hyperplane has to pass through the origin.

4 Informed Sampling Strategies

To recommend an item to a user u in a given context $\mathbf{s}$, we need to maximize the utility over all products $\mathbf{p}_i \in \mathbb{P}$. In many tasks with only a small set of items and a fast utility function, a complete enumeration, i.e. an evaluation of every possible product, may be feasible to find the best-rated item within $\mathcal{O}(|\mathbb{P}|)$. However, in real applications, the size of item set is very large, and an efficient technique is required for online recommendation. We present a variant of MCTS for reducing the search space and efficiently finding the (near-) optimal candidates.

Thus, we aim to minimize the computational costs for evaluating the utilities of the items while still obtaining a reasonable approximation of the optimum. One way to minimize the costs is to limit the number of examples under consideration which implies a trade-off between the utility value of the returned optimal product and the number of items considered, i.e., we strive for finding near-optimal products within a bounded number of items. For this purpose, we incrementally build up a search tree.

Structure of the Search Tree. Every product is characterized by a set of five categorical features: `color`, `gender`, `category`, `sub-category`, and `brand`. Figure 1 illustrates the search tree for this task which is constructed in a way that

each layer of the tree corresponds to one categorical feature. The actions/arcs between two layers correspond to values of this feature, e.g., setting the color of a product to `blue`. Therefore, a trajectory starting from the root node first chooses a value for feature 1, followed by a value for feature 2 and so on. At the leaf nodes of the tree, all five features have been assigned that leads to a complete item description, and can be used with the current user and context to query its utility value. We ensure that each leaf node corresponds to a real product. As an example, choosing action `shoes` in depth 3 sets the third feature of the product to `shoes`, making `shirts` an illegal action at depth 4.

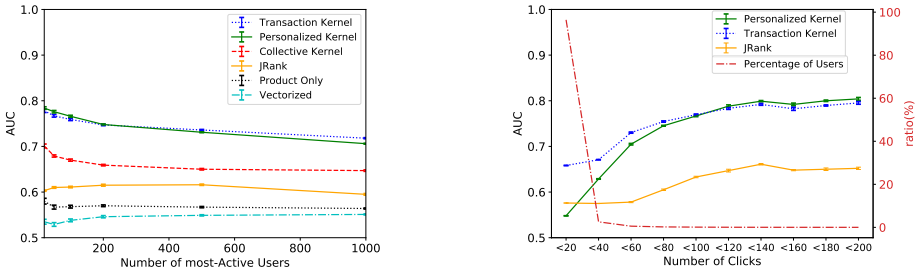
Monte-Carlo Tree Search. MCTS is an online search algorithm to explore the state space and find the optimal states for a given utility function. The key idea is to incrementally construct an asymmetric partial search tree, guided by the estimates for the encountered actions [14]. The tree is expanded deeper in branches with most promising actions, so that less time is spent on evaluating less promising ones. In our setting, the nodes correspond to product features and the leaves are the products to recommend. Thus, products with more promising features will be more likely to be sampled than products with uninteresting features. MCTS is an anytime algorithm, i.e., the algorithm can be stopped at any time and will provide the best result encountered up to that point.

The algorithm consists of four consequtive steps, which are iterated for each new example [2]. The *selection step* follows a path through the tree until a leaf is reached. The *expand step* adds a child of this node to the partial tree. Unless this leaf is already a terminal state in the search space (in our case a product with all features), Monte-Carlo sampling is used to sample a random terminal state below the current leaf (the *rollout step*). The value of this terminal state, in our case the utility value of the sampled product, is then propagated back through all nodes up to the root node (the *backpropagation step*). These backed up values in the nodes are further used in the next selection step.

Upper Confidence Bounds on Trees. In the selection step, the next node to expand can be selected in many different ways. We use Upper Confidence Tree (UCT) [14], which treats each node as a bandit problem. More precisely, in each node, it selects the action that maximizes the term

$$i = \arg\max_j \left(\bar{v_j} + 2 \cdot \lambda \sqrt{\frac{2 \ln n}{n_j}} \right), \tag{9}$$

where $\bar{v_j}$ is the average value propagated through the j-th node, n_j is the number of times a value has been propagated through this node, while n is the number of times a value has been propagated through its parent node. parameter λ trades off two terms in this formula, which correspond to exploitation (focusing on the best parts of the tree) and exploration (focusing on unexplored parts of the tree).



(a) (b)

Fig. 2. AUC values for (a) SVMs with different kernels vs. baseline, (b) various click distribution of users.

Algorithmic Modifications. Once a leaf node with high value is found and added to the partial search tree, it is very likely that MCTS will visit that node again to get a better estimation. For most applications of MCTS, this is a desired behavior. However, in our setting, a recently evaluated item does not necessarily need to be re-evaluated, since its value remains the same (deterministic setting). Hence, we remove actions/arcs from the search tree if all products reachable from this edge have already been evaluated in a previous iteration. In this way, we ensure that products are sampled at most once, but the search nevertheless focuses on products which match the important properties in the higher levels of the tree. To select the best product, we do not consider the most frequently visited actions, as the base MCTS algorithm would do, but keep track of the top encountered items with respect to the utility value. Although we only consider recommendations for the best product in this paper, the framework can be easily extended to do top-k recommendations.

5 Empirical Study

We conduct our experiments on a real-world dataset from Zalando, a large European online fashion retailer. The data contains pairwise preferences of ∼680k users on ∼16k items in ∼3.5m total transactions. A transaction occurs when a user prefers an item over another item, any other click data is ignored. Additionally, every item is characterized by a set of five categorical features: `color`, `gender`, `category`, `sub-category`, and `brand`. The evaluation of our proposed approach is twofold, firstly we study the performance of the personalized user preferences, and secondly, the efficiency of finding optimal items from the feature-based search tree is explored.

5.1 Performance of Preference Model

The pairwise preference model is obtained by training a support vector machine with the proposed transaction kernel. To assess the quality of the latter, we run

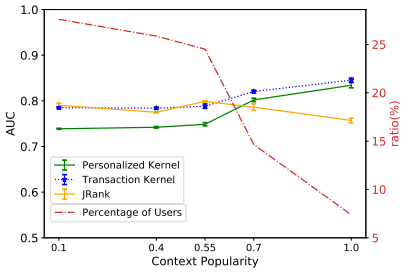


Fig. 3. AUC values for different context popularity.

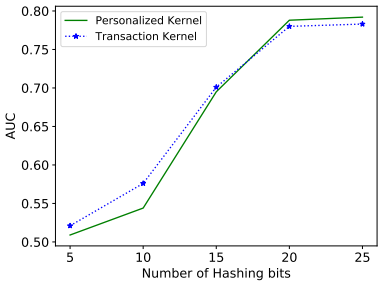


Fig. 4. Influence of different hash sizes in terms of AUC.

the SVM with different feature representations, i.e., kernels, and compare their performance. The preferences are further converted to both positive and negative examples to have a balanced dataset. Additionally, the features are represented as the unmodified *string* attributes of products for the hash function as shown in Table 1 with a hash size of 2^{17}. SVM trade-off parameters are optimized by model selection techniques.

The simplest feature representation in our experiment uses the five attributes of the products and discards user or context features. That is, $\psi(\mathbf{p}, u, \mathbf{s}) = \mathbf{p}$, where the features are one-hot encoded. We refer to this representation as "product only". A second baseline concatenates all available features into a single vector. Since the user ID is hardly an informative quantity, we replace it by clicking frequencies of item categories to form a feature vector for users. The frequencies are computed on historic data, and the final representation is given by $\psi(\mathbf{p}, u, \mathbf{s}) = [\mathbf{p}; \mathit{freq}_{cat}(u); \mathbf{s}]$. We refer to this representation as "vectorized". We include the collective kernel and the personalized kernel as special cases of the transaction kernel in our experiments. We also compare the performances with JRank [1], as another kernel-based recommendation approach which is based on collaborative filtering view to predicts ordinal ratings; this corresponds to learning a binary ranking in our scenario.

Figure 2 (a) shows AUC values obtained by a 10-fold cross-validation for the different models. The users on the x-axis are ordered from left to right according to their click frequencies. The results clearly demonstrate that the information encoded in the baselines "product only" and "vectorized" is not sufficient to accurately learn the preferences. The poor performance of JRank is caused by the sparsity of the data as well as cold start situations. JRank tries to remedy cold start issues by incorporating attributes of users and items into the model; however, compared to including the short-term context as in our model, there is only little to incorporate for JRank. Furthermore, the correlation kernel in JRank depends on collaborative information extracted from the user-item matrix which is too sparse in the problem to capture accurate correlations between users and items. The collective kernel alone also does not perform well. The reason for this

lies in the choice of the users. Since all users expressed many preferences, they left enough data for the personalized kernel to capture their characteristic traits. However, at about rank 200, the performance of the transaction kernel increases over the personalized kernel as the collective kernel kicks in. The users with 200 to 1000 clicks clearly benefit, if only slightly, from the inclusion of the collective model into the transaction kernel.

To illustrate this effect, consider Figure 2 (b) which uses all data. The x-axis shows the different numbers of clicks of the users together with the distribution. Simultaneously, the figure shows the AUC of the transaction and the personalized kernel (y-axis on left side). In terms of AUC, users who have about 120 clicks or more are better off with a purely personalized model that is only trained on their data alone. However, the red curve shows that these users are only a vanishing minority of all users. The distribution of clicks clearly follows an exponential law where the majority of users have only one or two dozens of clicks. For them, the transaction kernel leverages the collective kernel so that the majority benefits, even though they have only little data to share. On the other side of the scale, the heavy hitters do not loose too much in terms of AUC if the transaction kernel is used. This renders that the proposed approach provides the best representation in this study. Note that the two left-most points of the figure involve data at large scales and the first cross validation fold for JRank took more than $\approx$30 days. We thus resort to showing only the results of this first fold.

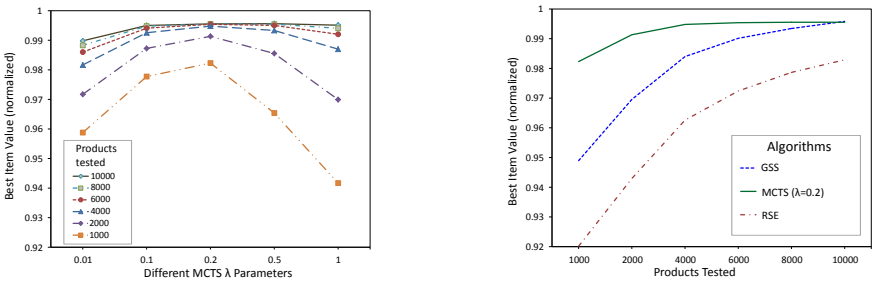
Figure 3 draws a similar picture for context popularity instead of user clicks. We randomly choose a smaller subset of data for this experiment to evaluate the baseline in a reasonable amount of time, which leads to an overall higher AUC in all the approaches. The figure confirms that the more popular the context, the better the performance of the transaction kernel. In addition, popular contexts are rare, and again the majority of users create new and unseen contexts. However note that the transaction kernel clearly outperforms the personalized kernel for all contexts. JRank does not rely on any context related information and is more or less unaffected by context popularity.

We also investigate the effect of the size of the hashing function. Figure 4 shows the results for data from the 200 most active users. The more bits are used, the larger the resulting hashed space. Smaller numbers of bits lead to collisions that introduce noise into the representation as the mapping is no longer one-to-one but one-to-many.

5.2 Performance of Informed Sampling

We compare the performance of our MCTS-based sampling with other search strategies: Random Subset Exhaustion (RSE) and Greedy Stochastic Search (GSS) as the baselines. RSE is a simple way to approximate the optimal element of a countable set without taking any structure into account. Given a fixed limit on the number of products that can be tested, it takes a random sample of the given size and exhaustively determines the best discovered item.

GSS explores the search tree in a greedy fashion. We first randomly initialize a product, and then explore a stochastic neighborhood of this product. This



(a) (b)

Fig. 5. Performance of MCTS (a) w.r.t. different parameters, and (b) compared to baseline methods.

neighborhood is formed by sampling a fixed number of h products, where half of them are randomly selected among those that differ only in brand (distance $d = 2$ edges in the tree of Figure 1), one quarter of the products differ in brand and subcategory ($d = 4$), and, in general, $1/d$ percent of the products in the neighborhood are randomly selected among those with distance d. Then, all products among the h products in this neighborhood are evaluated and the search continues with the best one among those.

We run the experiments on several user-context pairs, where we randomly select three existing users and one new user. For the context, we randomly select 50 items which leads to 200 different settings. The results of each context-user pair and parameter setting are averaged over 50 runs for MCTS and GSS. For RSE, we show the average over all possible subsets. We first evaluate the performance of the proposed method for various values of λ that we choose from $\{0.01, 0.1, 0.2, 0.5, 1\}$. Figure 5 (a) shows that the value of $\lambda = 0.2$ achieves the best result for different numbers of tested products.

We further evaluate the performance of the MCTS approach compared to the baselines. Figure 5 (b) shows the average value for the best found product for different numbers of tested items. The results confirm that informed sampling via MCTS outperforms RSE for the same number of products for all maximum sample sizes. The advantage can be observed for all settings of the parameter λ , but the magnitude of the advantage varies. The performance of GSS lies between RSE and MCTS until $10,000$ products, and then starts to slightly outperform MCTS with an average value of over 99.5%. The parameter value of $\lambda = 0.2$ performs the best over all numbers of items considered. It is not surprising that this value favors exploitation, since our algorithmic modifications of MCTS already enforces some exploration.

In Figure 6, we show a more detailed analysis for a single product (hand-picked as a representative case). The values of the best found item are shown for different users and different parameter settings. For all selected users, MCTS with $\lambda = 0.2$ finds a better product within $2,000$ products than RSE does within

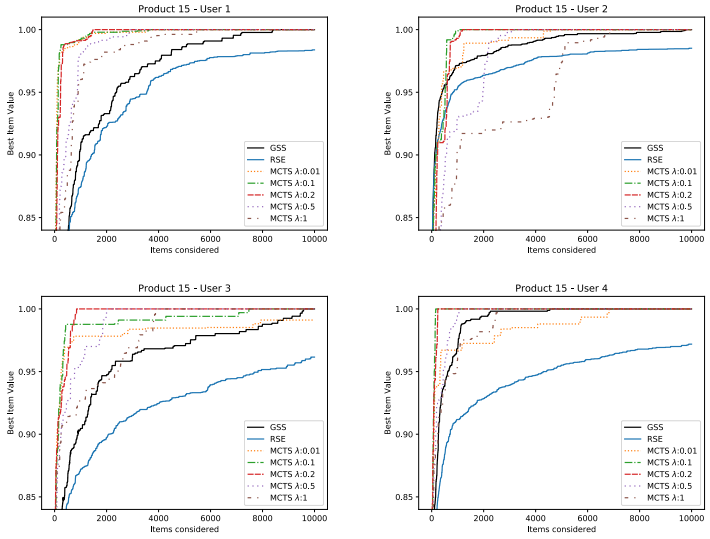


Fig. 6. Given one product and four users (user 4 is a new user), different MCTS parameters, GSS and RSE are tested.

10,000 products. There are extreme cases like for User 4, a new user without training data, where MCTS is able to find the best-rated item very quickly. For User 2, a bad choice for MCTS, the parameter λ shows a worse performance than RSE, but this is a rare case. In summary, MCTS is able to find near-optimal products using a considerably lower number of samples than GSS or RSE. However, this reduction in the number of tested products comes with higher computational costs. Therefore, the choice of sampling strategy depends on the problem at hand.

6 Conclusion

In this paper, we presented an effective and efficient preference-based learning approach to model personalized interests of users in contextual settings. We devised transaction kernels from pairwise preference data that combine theories from tensor products with hashing functions to capture individual as well as collective user preferences. The kernel functions were used in training a preference model via support vector machines to predict the utility of various products for a given user and context. Subsequently, we proposed a variant of Monte Carlo tree search method for efficiently retrieving near-optimal items for online recommendation purposes. Empirically, on a real-world transaction dataset, both the preference models as well as the search tree exhibited excellent performance over baseline approaches, in particular in cases where only a small number of products could be sampled.

References

1. Basilico, J., Hofmann, T.: Unifying collaborative and content-based filtering. In: Proceedings of the 21st International Conference on Machine Learning (ICML-04). p. 9. ACM (2004)
2. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 1–43 (March 2012)
3. Cao, B., He, L., Kong, X., Philip, S.Y., Hao, Z., Ragin, A.B.: Tensor-based multi-view feature selection with applications to brain diseases. In: Proceedings of the IEEE International Conference on Data Mining (ICDM-14). pp. 40–49 (2014)
4. Cao, B., Kong, X., Yu, P.S.: A review of heterogeneous data mining for brain disorder identification. Brain Informatics **2**(4), 211–233 (2015)
5. Chapelle, O., Keerthi, S.S.: Efficient algorithms for ranking with SVMs. Information retrieval **13**(3), 201–215 (2010)
6. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: International Colloquium on Automata, Languages, and Programming. pp. 693–703. Springer (2002)
7. Dullemond, K., Peeters, K.: Introduction to Tensor calculus. University of Heidelberg (2010), http://www.e-booksdirectory.com/details.php?ebook=9967
8. Fürnkranz, J., Hüllermeier, E. (eds.): Preference Learning. Springer-Verlag (2010)
9. Gaudel, R., Sebag, M.: Feature selection as a one-player game. In: Fürnkranz, J., Joachims, T. (eds.) Proceedings of the 27th International Conference on Machine Learning (ICML-10). pp. 359–366. Omnipress, Haifa, Israel (2010)
10. Gemmis, M.d., Iaquinta, L., Lops, P., Musto, C., Narducci, F., Semeraro, G.: Learning preference models in recommender systems. In: Fürnkranz and Hüllermeier [8], pp. 387–407
11. Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: Proceedings of the 8th IEEE International Conference on Data Mining (ICDM'08). pp. 263–272. Ieee (2008)
12. Joachims, T.: Optimizing search engines using clickthrough data. In: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 133–142. ACM (2002)
13. Kamishima, T., Kazawa, H., Akaho, S.: A survey and empirical comparison of object ranking methods. In: Fürnkranz and Hüllermeier [8], pp. 181–201
14. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Proceedings of the 17th European Conference on Machine Learning (ECML'06). pp. 282–293. Springer-Verlag, Berlin, Heidelberg (2006)
15. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8) (2009)
16. Li, L., Chu, W., Langford, J., Schapire, R.E.: A contextual-bandit approach to personalized news article recommendation. In: Proceedings of the 19th International Conference on the World Wide Web (WWW-10). pp. 661–670. ACM (2010)
17. Liebman, E., Khandelwal, P., Saar-Tsechansky, M., Stone, P.: Designing better playlists with monte carlo tree search. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-17). pp. 4715–4720 (2017)
18. Oyama, S., Manning, C.: Using feature conjunctions across examples for learning pairwise classifiers. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD-17). pp. 322–333. Springer (2004)

19. Pham, N., Pagh, R.: Fast and scalable polynomial kernels via explicit feature maps. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 239–247. ACM (2013)
20. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on the World Wide Web (WWW-01). pp. 285–295. ACM (2001)
21. Schölkopf, B., Herbrich, R., Smola, A.J.: A generalized representer theorem. In: Proceedings of the International Conference on Computational Learning Theory (COLT-01). pp. 416–426. Springer (2001)
22. Shi, Q., Petterson, J., Dror, G., Langford, J., Smola, A., Strehl, A., Vishwanathan, S.: Hash kernels. In: Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS-09). pp. 496–503. JMLR, Clearwater Beach, Florida, USA (2009)
23. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
24. Smalter, A., Huan, J., Lushington, G.: Feature selection in the tensor product feature space. In: Proceedings of the 9th IEEE International Conference on Data Mining (ICDM'09). pp. 1004–1009. IEEE (2009)
25. Tavakol, M., Brefeld, U.: A unified contextual bandit framework for long-and short-term recommendations. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD-17). pp. 269–284. Springer (2017)
26. Tavakol, M., Brefeld, U.: Factored MDPs for detecting topics of user sessions. In: Proceedings of the 8th ACM Conference on Recommender Systems. pp. 33–40. ACM (2014)
27. Vanchinathan, H.P., Nikolic, I., De Bona, F., Krause, A.: Explore-exploit in top-n recommender systems via gaussian processes. In: Proceedings of the 8th ACM Conference on Recommender Systems. pp. 225–232. ACM (2014)
28. Vembu, S., Gärtner, T.: Label ranking algorithms: A survey. In: Fürnkranz and Hüllermeier [8], pp. 45–64
29. Wang, C., Blei, D.M.: Collaborative topic modeling for recommending scientific articles. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 448–456. ACM (2011)
30. Wang, Y., Tung, H.Y., Smola, A.J., Anandkumar, A.: Fast and guaranteed tensor decomposition via sketching. In: Advances in Neural Information Processing Systems. pp. 991–999 (2015)
31. Weinberger, K., Dasgupta, A., Langford, J., Smola, A., Attenberg, J.: Feature hashing for large scale multitask learning. In: Proceedings of the 26th International Conference on Machine Learning (ICML-09). pp. 1113–1120. ACM (2009)
32. Zelenko, D., Aone, C., Richardella, A.: Kernel methods for relation extraction. Journal of Machine Learning Research **3**(Feb), 1083–1106 (2003)